multiplier operating in Cartesian coordinates requires a minimum of three general multiplications of real numbers.

(1) $X[k] = \Sigma_{n=0 \text{ to } N-1} x[n] w[n,k]$ for $k = 0,1,\ldots,N-1$

When the weight w[n,k] is equal to $\exp(\{-j 2 \pi n k / N\})$, the transform of equation (1) is the discrete Fourier transform or DFT. When the weight w[n,k] is equal to $\exp(\{j 2 \pi n k / N\})$ the transform of equation (1) is the inverse DFT scaled by a factor N.

Signal processing transforms such as the DFT and the inverse DFT are implemented in a variety of technologies for a variety of applications. In some applications, a signal processing transform is computed repeatedly, and it is desirable to re-use circuitry or operation sequences. In real-time applications, the time available for computation is limited, so the transform should be computed quickly. Regardless of the application, there is a cost associated with the implementation. For economic and other reasons, it is useful to have low-complexity transforms and transform implementations.

DESCRIPTION - REDUCED-COMPLEXITY MULTIPLICATION TECHNIQUES

One way to reduce the cost of computing a signal processing transform that uses large numbers of multiplication operations is to reduce the cost of individual multiplication operations. There are many prior art examples of such techniques. Most are based on the principle of eliminating steps in the multiplication process that are not necessary or on the principle of replacing one or more necessary but high-cost steps with lower-cost but equivalent alternatives.

As a very simple example, one might "multiply" a number by 2 by adding the number to itself. A lower-cost addition operation can replace a higher-cost multiplication operation. Similarly, one might multiply a number by 3 by adding the number to itself to produce a first sum, and then adding the number to that first sum.

The procedures just above for multiplying a number by 2 and by 3 can be made even less costly, depending on the finite-precision numeric format in which the number is represented. Many binary representations have a bits which represent power-of-two numeric values. Shifting some of the bits in such representations can implement scaling by power-of-two factors. For instance, shifting bits one place can implement scaling by a factor of 2, or by a factor of 1/2, depending on the format and the direction of the shift.

One may be able multiply a number by 2 by shifting bits in that number's representation in a suitable finite-precision numeric format. This shifting might involve physical movement of bit values from one storage element in a storage unit to another element in the same storage unit, which could be a low-cost operations. Alternatively, it might involve a circuit design in which storage elements in one storage unit are wired to storage elements in a storage unit of a next processing stage which correspond to shifted storage locations.

One may be able to multiply a number by 3 by shifting bits in a copy of its representation to double the value of the copy, followed addition of the shifted copy to the original representation of the number. If shifting has lower cost than addition, this technique for multiplication by 3 may have lower cost than the technique described above.

Another tool that enables low-cost multiplication is negation. In signed binary representations, there is a bit which indicates whether a number is positive or negative. Multiplication by -1 can be implemented by flipping the value of this sign bit. Similarly, negation of a number in a binary twos complement representation can be implemented by flipping all the bits of the representation and adding a constant to the result.

Low-cost operations such as addition, shifting, and element-changing (bit-flipping, in the case of binary storage elements) enable low-cost multiplication for particular number values in particular finite-precision numeric formats. The lowest-cost

implementations for individual product computation are constant multipliers in which all control and operational steps necessary in a non-constant multiplier have been eliminated.

Constant multipliers have been proposed for implementing transforms such as discrete Fourier transforms and discrete cosine transforms. For useful transform sizes, these transforms use a large number of multiplication operations. In applications, the transforms are often computed repeatedly. Also, and most importantly, the transforms have fixed weights that are known in advance.

The cost of a particular constant multiplier may depend on the number value of the constant, the representation of the constant, and the finite-precision numeric format of the representation. However, the use of a constant multiplier does not require a particular number value or finite-precision numeric format, only that one of the two numbers being multiplied is the constant. The transforms for which constant multipliers have been proposed are those which have fixed, known weights.

The concept of replacing necessary but costly steps with alternative lower-cost steps appears in the constant multipliers proposed by US Patent 4,868,778 issued to J.E. Disbrow on September 19, 1989 and by US Patent 5,841,684 issued to K. Dockser on November 24, 1998. Analogous to the technique discussed above for multiplying a number by 3 using a shift operation and an addition operation instead of two addition operations, these patents propose computation of a partial product, shifting of that partial product, and addition. This saves addition operations when there are repeated patterns in a particular representation of a constant. The partial products are intermediate terms which are used in computing the desired final product.

The reduced-complexity multiplication techniques described above rely on either the special properties of number values, on special properties of number representations in particular finite-precision numeric formats, or on both. They consider multiplication of one number by another number. However, they do not take into account the fact that